,,,,
The University of Sydney
School of Computer Science
Dr Suranga Seneviratne
Senior Lecturer - Security

THE UNIVERSITY OF SYDNEY

Hashes, MACs, and Signatures

Recommended Reading

Cryptography and Network Security (7^{th} Edition - William Stallings)

- **Chapter 11** - Cryptographic Hash Functions
- **Chapter 12** - Message Authentication Codes
- **Chapter 13** – Digital Signatures

1 Hash Functions

A hash function is a one-way function that accepts a variable-length block of data M as input and produces a fixed-size hash value $h = H(M)$. The output of a hash function is referred to by different names depending on the context, such as digest, hash, hash value, checksum, CRC, and more. Given that the input space of a hash function is significantly larger than its output space, collisions are inevitable occurrences. A hash function deemed as "good" possesses the characteristic whereby the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and emulate randomness.

Following is a list of commonly used hash functions.

- Cyclic Redundancy Check (CRC) (CRC-16, CRC-32, . . .)
- MD5 (MD - Message Digest)
- SHA1 SHA-2, SHA-3 (SHA - Secure Hash Algorithm)
- RIPEMD160

From the above examples, CRC was not designed for cryptography, and MD5 and SHA-1 were found to be insufficient, even though they were designed for cryptography.

The principle objective of hash functions in cryptography is data integrity. At a conceptual level, the process operates as follows: There exists a predefined function (hash function) capable of generating a checksum for a message. When the sender wishes to transmit a message, they first pass it through this hash function to obtain its hash value. Subsequently, the sender can encrypt the original message using an encryption scheme to safeguard its confidentiality. Finally, the sender dispatches both the encrypted message and its corresponding hash to the receiver. Upon receiving the transmission, the

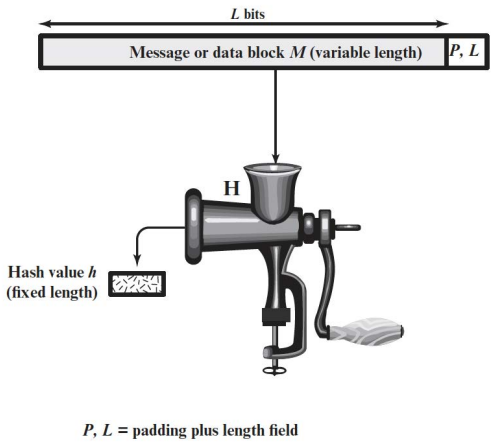


Figure 1: Cryptographic hash function
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

receiver decrypts the message and recalculates its hash. They then compare this recalculated hash with the one received. If the two hash values match, the receiver can confidently ascertain that the message has remained unaltered during its transit. In this scenario, it is important that the following conditions are met.

a. An attacker cannot deduce the original message from the hash, as this would compromise confidentiality

b. An attacker must not be able to find a plaintext that results in the same hash. Otherwise, the attacker can replace the plain text without changing the hash value, rendering the alteration undetectable to the receiver.

Hash functions that satisfy the above criteria are referred to as ***cryptographic hash functions***. Figure 1 illustrates the fundamental operation of a cryptographic hash function. Accordingly, the input of variable length is padded to an integer multiple of a predetermined fixed length (known as the block size). The padding incorporates the length of the original message in bits as part of its procedure which serves as a security measure, augmenting the challenge for potential attackers to generate an alternative message with an identical hash value. Subsequently, the message undergoes hashing, producing the hash value h.

1.1 Applications of Cryptographic Hash Functions

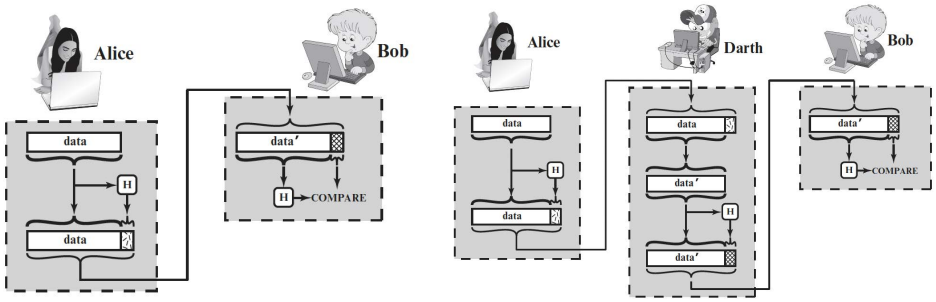
Cryptographic hash functions are widely used in various security applications and Internet protocols. Exploring the range of tasks they handle can shed light on the specific requirements and security implications they entail.

1.1.1 Message Authentication

Message authentication serves as a mechanism to validate the integrity of a message. Often, there is an additional requirement for the authentication process to confirm the sender's claimed identity. When employing a hash function for message authentication, the resulting value is commonly termed a *message digest*.

Figure 2a illustrates the basic use of hash functions for integrity. The sender generates a hash value based on the message and sends both the hash value and the message. Upon reception, the receiver

conducts the same hash computation on the message bits and compares the resultant value with the transmitted hash value. In the event of a disparity, the receiver infers that either the message itself or the hash value has undergone alteration. However, this method is vulnerable to person-in-the-middle attacks as depicted in Figure 2b. Accordingly, an attacker like Darth who intercepts Alice's message can replace the original message and alter the message digest to match the new message. Bob who receives the altered data with the new hash value will not be able to detect the change. This problem is commonly referred to as origin authentication and is closely related to integrity. To prevent such attacks, the message digest generated by Alice must be protected.



(a) Hash function for integrity (b) Person-in-the-middle attack

Figure 2: Message authentication
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

Message Authentication Codes (MACs) which are also known as **keyed hash functions** are a common way of achieving message authentication (i.e. integrity and origin authentication). MACs rely on a shared secret key between the communicating parties. They generate a hash value, known as the MAC, by incorporating both the message and the secret key, which is then transmitted alongside the message. To ensure the message's integrity, the MAC function can be applied to the message, and the outcome compared with the received MAC value. An attacker attempting to modify the message would be unable to alter the corresponding MAC value without access to the secret key. Moreover, the verifying party can ascertain the identity of the sender, as only the parties privy to the secret key possess this capability. MACs are discussed in more detail in Section 2.

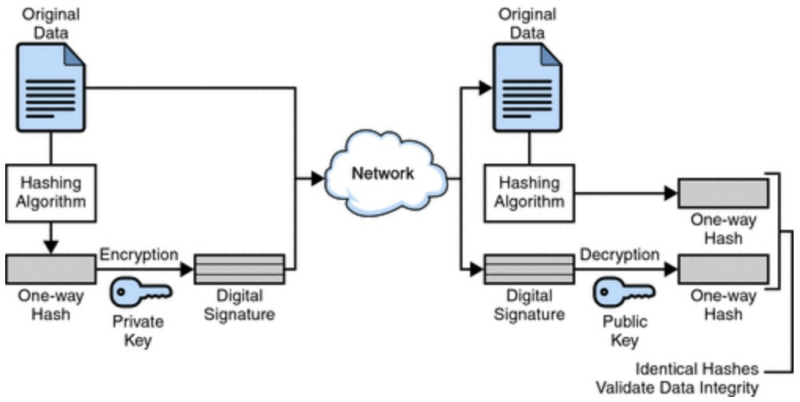
1.1.2 Digital Signatures

Digital signatures operate similarly to MACs but rely on public key cryptography rather than a shared secret key. As illustrated in Figure 3, digital signatures encrypt the hash value of a message using the sender's private key which is then transmitted alongside the message. Anyone who knows the sender's public key can verify the integrity of the message associated with the digital signature and confirm that the message came from the expected sender. In this scenario, any potential attacker aiming to modify the message and/or hash would require access to the sender's private key. Digital signatures are discussed in detail in Section 3.

In addition to MACs and digital signatures, cryptographic hash functions are used in many other applications including one-way password files, intrusion detection, virus detection and pseudorandom number generator (PRNG).

1.2 Security Requirements for Cryptographic Hash Functions

Preimage: A specific input value (or set of input values) that produces a particular output value when passed through a given hash function. For a hash value $h = H(x)$, x is referred to as the preimage of h.



Source: https://docs.oracle.com/cd/E19424-01/820-4811/aakfx/index.html

Figure 3: Digital Signatures

Collision: If two different inputs result in identical hash values when processed by the same hash function, it constitutes a collision. i.e. if $H(a) = H(b), a \neq b$, this is a collision.

Formally the security requirements of cryptographic hash functions are as listed below.

1. Preimage resistance
2. Second Preimage resistance
3. Collision resistance

1.2.1 Preimage Resistance

Preimage resistance specifies that an attacker must not be able to find the preimage (the input) value by observing the hash value. More formally,

> Given a randomly chosen y from H's range of output values, it is computationally infeasible to find an x such that $H(x) = y$.

Note: This property does not guarantee the impossibility of finding a preimage for a given output (hash value), as brute-forcing remains a potential method. However, it mandates that discovering such values should be computationally infeasible.

This property is important if the confidentiality of the original message is required since if preimage resistance is violated, an attacker can infer the original message by observing the hash value.

1.2.2 Second Preimage Resistance

Second preimage resistance specifies that if the attacker is given a preimage (input), they cannot find a second preimage that would have the same output hash value. More formally,

> Given a randomly chosen x, it is computationally infeasible to find an x', $x \neq x'$ such that $H(x) = H(x')$.

If this property is compromised, it would enable an attacker to substitute the original message with a forged one that yields the same hash value.

1.2.3 Collision Resistance

Collision resistance specifies that an attacker is unable to find any two input values that have the same output hash value. More formally,

> It is computationally infeasible to find a pair (x, x'), $x \neq x'$ such that $H(x) = H(x')$.

Note: Collision resistance implies second preimage resistance.

In certain scenarios, compromising a cryptographically secure hash function merely involves discovering collisions, a task more feasible than one might expect. This is due to a phenomenon known as the *birthday attack*.

Birthday Attack

If n random people are invited to a party, there is more than a 50% chance of two people having the same birthday if $n \geq 23$. This idea can be generalized as follows.

> **Birthday Attack**: If random variables are chosen from a uniform distribution in the range 0 through $N - 1$, then the probability that a repeated element is encountered exceeds 50% after $\sqrt{N}$ choices have been made.

Once applied to hash functions, birthday attack can be interpreted as follows. For a hash function that has an output length of n (in bits), more than $\sqrt{2^n}$ attempts will have more than 50% chance of producing a collision. For instance, for a hash function with output length 64, 2^{32} attempts have a 50% chance of producing collisions. Hence, all cryptographic hash functions must have sufficient output length. Currently, the accepted minimal output length of a cryptographic hash function is $n \geq 160$ bits ($\sqrt{2^{160}} = 2^{80}$).

Figure 4 illustrates the relationships among the three resistance properties. A hash function that exhibits collision resistance also exhibits second preimage resistance; however, the converse is not always accurate. It is plausible for a function to be collision resistant but not preimage resistant, and vice versa. Similarly, a function may be preimage resistant but not second preimage resistant, and vice versa. Furthermore, Table 1 shows the resistant properties required for various hash function applications.

	Preimage Resistant	Second Preimage Resistant	Collision Resistant
Hash + digital signature	Yes	Yes	Yes[1]
Intrusion detection and virus detection		Yes	
Hash + symmetric encryption			
One-way password file	Yes		
MAC	Yes	Yes	Yes[1]

Table 1: Hash Function Resistance Properties Required for Various Data Integrity Applications

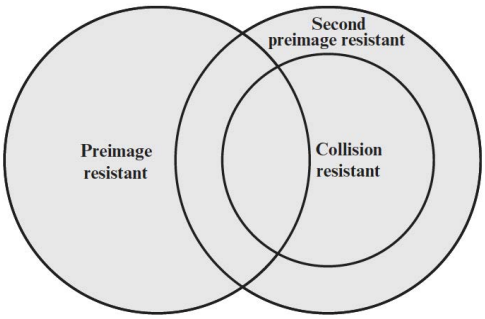


Figure 4: Relationship Among Hash Function Properties
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

1.2.4 Pseudo-randomness

Pseudo-randomness refers to the property of a sequence of numbers that appears random but is generated by a deterministic process. While pseudo-randomness has not been traditionally listed as a requirement of cryptographic hash functions, it is implied that hash functions must output pseudo-random numbers. Cryptographic hash functions find widespread application in tasks such as key derivation and pseudorandom number generation. In message integrity applications, the effectiveness of these functions relies on their output exhibiting the properties of randomness. Therefore, it is recommended to verify that a given hash function indeed generates output that appears pseudo-random.

1.3 Secure Hash Algorithm

In recent years, the Secure Hash Algorithm (SHA) has emerged as the most prevalent hash function. Developed by the National Institute of Standards and Technology (NIST), SHA was introduced as a federal information processing standard (FIPS 180) in 1993. However, the initial iterations, SHA-0 and SHA-1 (with 160-bit output), have fallen out of favor due to their susceptibility to collisions. Then in 2002, NIST produced a revised version that is referred to as SHA-2 with three variants based on the output sizes SHA-256, SHA-384, and SHA-512. In 2015, NIST announced the latest member of the SHA family, SHA-3 with variants SHA3-224, SHA3-256, SHA3-384, and SHA3-512. Table 2 summarizes the parameters of different SHA versions.

Algorithm	Message size	Block size	Word size	Digest size
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Table 2: Comparison of SHA parameters

[1] Resistance required if an attacker is able to mount a chosen message attack.

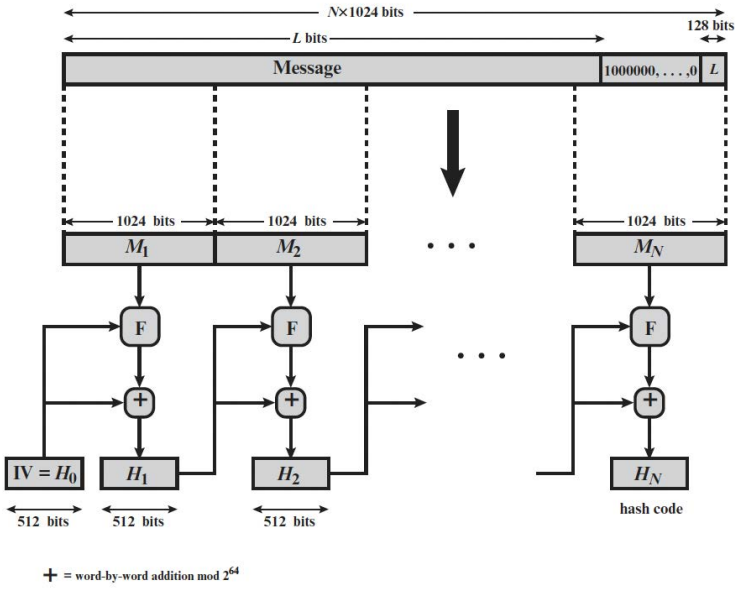


Figure 5: Message Digest Generation Using SHA-512
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

1.3.1 SHA-2

For example, SHA-512, in the SHA-2 family, takes an input message with a maximum length of less than 2^{128} bits and produces a 512-bit message digest after processing the input in 1024-bit blocks as illustrated in Figure 5. F is called as the compression function and its operation is shown in Figure 6.

2 Message Authentication Code (MAC)

Message authentication refers to the process of verifying that a received message comes from the said source and has not been altered. Any mechanism for message authentication or digital signatures consists of two levels of functionality. At the lower-level, there exists a function responsible for generating an authenticator: a value utilized to authenticate a message. This lower-level function is subsequently incorporated as a primitive component within a higher-level authentication protocol, enabling a recipient to authenticate the legitimacy of a message. The functions that can be used to generate an authenticator fall under three classes as follows.

- **Hash function:** As illustrated in Figure 2b, using hash functions alone may not provide message authentication as unprotected hash values are susceptible to person-in-the-middle attacks.
- **Message encryption:**
  - **Symmetric encryption**
    As illustrated in Figure 7a, encrypting the message with the shared key not only ensures confidentiality but also allows B to confirm that the message originated from A, as the secret key is known only to A and B. However, this assurance does not extend to all scenarios. For instance, an attacker can modify the ciphertext's bit pattern even without the knowledge of the secret key. If the original message was indeed a bit sequence, the receiver cannot detect the alteration because the modified ciphertext still decrypts into a valid bit sequence.
  - **Public-key encryption**

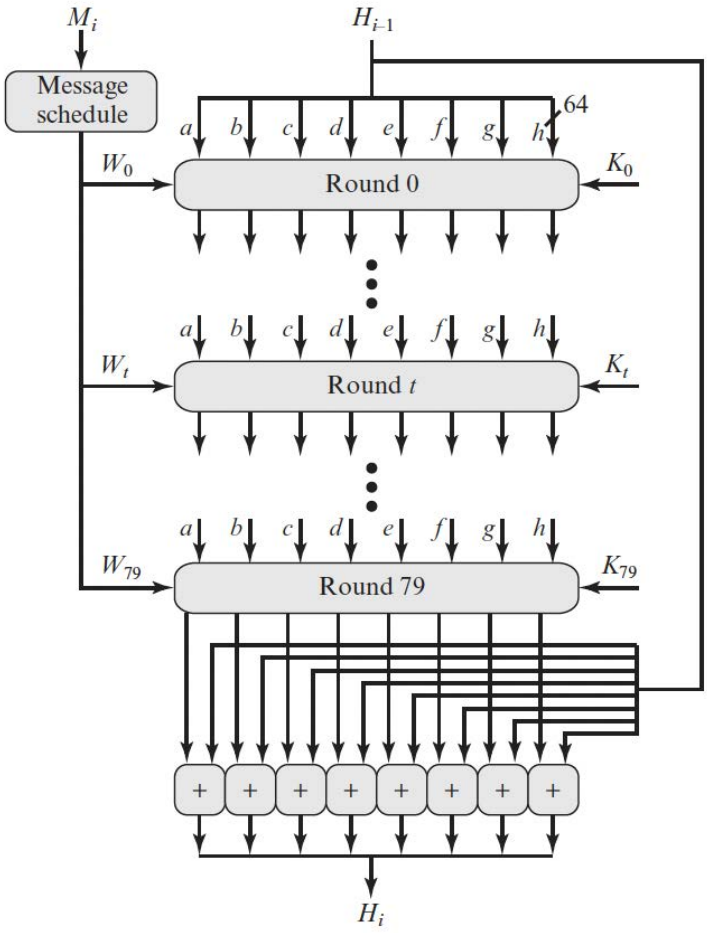
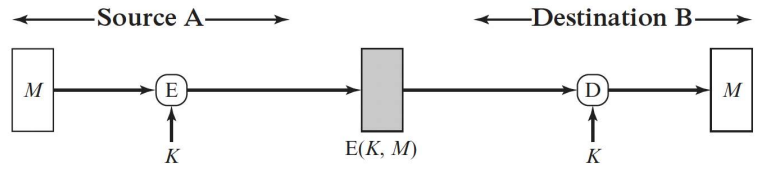


Figure 6: SHA-512 Processing of a Single 1024-Bit Block
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

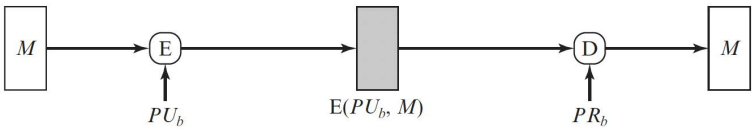
The straightforward use of public-key encryption as shown in Figure 7b ensures confidentiality but lacks authentication. This is because anyone can encrypt a forged message using B's public key and substitute it for the original message without B detecting the alteration. To provide authentication, A can use their private key to encrypt the message, and B can use A's public key to decrypt as illustrated in Figure 7c (Note that this method does not provide confidentiality as anyone can use A's public key to access the plain text). This allows B to confirm that the message originated from A, as the secret key is known only to A and B. However, this approach also has the same limitation as encrypting with symmetric keys: an attacker can modify the message without knowing A's private key, potentially going undetected by B if the altered ciphertext corresponds to a legitimate message.

To ensure both confidentiality and authentication, A can encrypt M first using its private key to create a digital signature (for authentication), and then encrypt it using B's public key to maintain confidentiality as shown in Figure 7d. However, this approach comes with a drawback: the complex public-key algorithm needs to be executed four times instead of two in each communication.

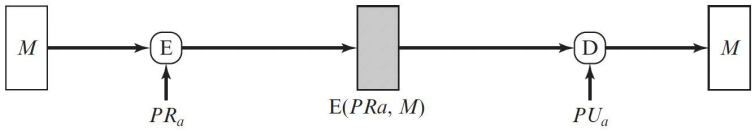
- **Message authentication code (MAC)**



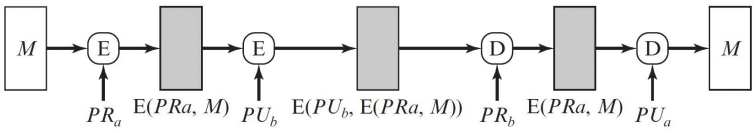
(a) Symmetric encryption: confidentiality and authentication



(b) Public-key encryption: confidentiality



(c) Public-key encryption: authentication and signature



(d) Public-key encryption: confidentiality, authentication, and signature

Figure 7: Basic Uses of Message Encryption
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

MAC is a special tag of fixed size, generated using a secret key shared between the communicating parties, that is appended to the original message. MACs are also known as cryptographic checksum.

$$MAC = C(K, M)$$

M = input message, C = MAC function, K = shared key, MAC = message authentication code.

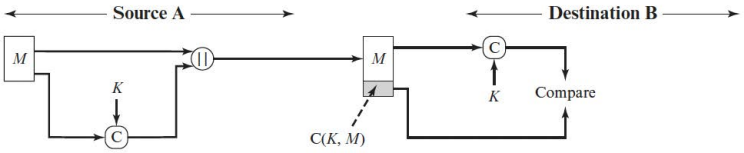


Figure 8: Message Authentication Code
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

The original message, along with the MAC generated as shown in Figure 8, is transmitted to the intended recipient. Upon receiving the message, the recipient performs the same calculation using the shared secret key to generate a new MAC. If the MACs match, assuming the secret key is known only to the sender and receiver, the following is true.

1. The recipient is assured that the message has not been altered. Any alterations made by an attacker to the message, without modifying the MAC, will result in a mismatch between the received and calculated MACs. Since the attacker does not possess the secret key, they cannot produce a valid MAC corresponding to the altered message.
2. The recipient can be assured of the message's origin, as no other party has access to the secret key required to generate the valid MAC.
3. Additionally, if the message includes a sequence number (e.g., used in protocols like HDLC, X.25, and TCP), the recipient can verify the sequence's integrity. An attacker cannot successfully alter the sequence number without detection.

The process depicted in Figure 8 does not provide confidentiality. Confidentiality can be provided by performing message encryption either after or before the MAC algorithm.

2.1 Requirements for Message Authentication Codes

The intuition behind message authentication codes is that given the MAC t for a message m, an attacker should not be able to generate a valid MAC for a different message. Accordingly, a function used as a MAC should satisfy the following requirements.

1. MAC function must have resistance properties of a cryptographic hash function.
2. Must be computationally infeasible to predict a correct MAC t' for a message $m' \neq m$ even when allowed to know any other combinations of (m, t).

A cryptographic hash function fulfills the first requirement. Incorporating a secret s would address the second requirement provided that the resistance properties are upheld.

Note: While there are MACs based on the use of a symmetric block cipher, in this course we only focus on the MACs derived from a cryptographic hash function.

2.2 HMAC (Hash-Based Message Authentication Code

While there have been several methods proposed regarding the integration of a secret key into an established hash algorithm, the most widely used approach is HMAC [**?**, **?**]. HMAC, published as RFC 2104, has been designated as the obligatory MAC for IP security and is employed in various internet protocols, including SSL. Furthermore, HMAC has been ratified as a NIST standard (FIPS 198).

HMAC can be expressed as follows.

$$HMAC(K, M) = H[(K \oplus opad) \parallel H[(K \oplus ipad) \parallel M]]$$

- H can be any cryptographic hash function (e.g., MD5, SHA-1)
- K is the shared secret key
- M is the input message
- *opad* and *ipad* are publicly known constant bit strings that are required for the security proof of HMAC to hold

The structure of HMAC is illustrated in more detail in Figure 9 where,

H = embedded hash function	M = message input to HMAC
b = number of bits in a block	L = number of blocks in M
K = secret key	K^+ = K padded with zeros on the left to match b
IV = initial value input to hash function	$Y_i = i^th$ block of M, $0 \leq i \leq (L-1)$
ipad = 00110110 (36_{16}) repeated b/8 times	opad = 01011100 ($5C_{16}$) repeated b/8 times
n = length of hash code produced by H	

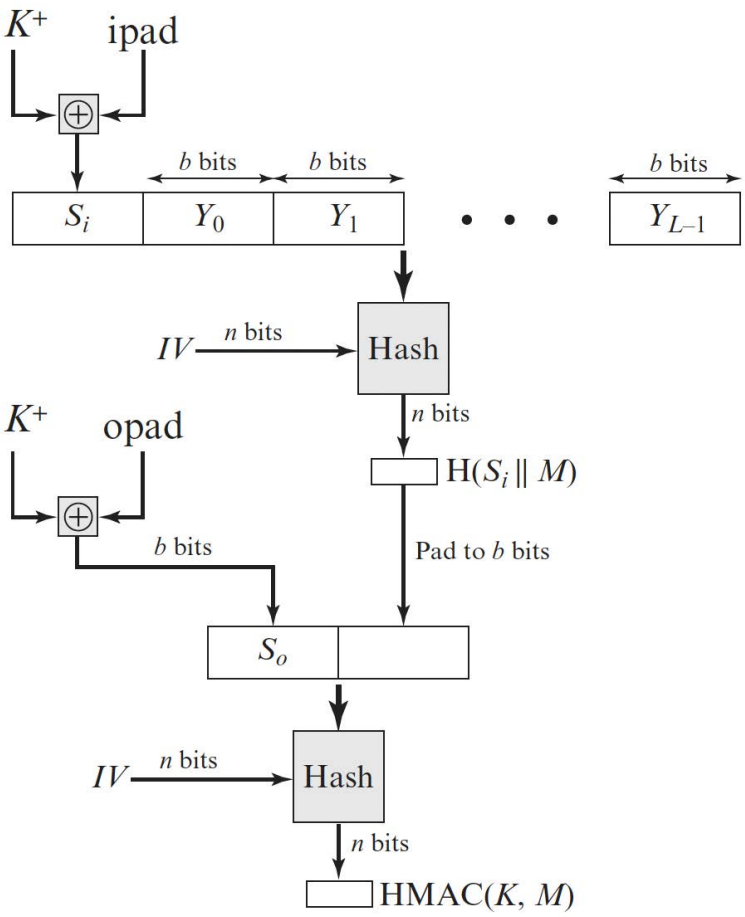


Figure 9: HMAC Structure
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

HMAC algorithm in Figure 9 can be described as follows.

1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and $b = 512$, then K will be appended with 44 zeroes).
2. XOR K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_0.
6. Append the hash result from step 4 to S_0.
7. Apply H to the stream generated in step 6 and output the result.

2.3 Security of HAMC

The security of any MAC function based on an embedded hash function depends in some way on the cryptographic strength of the underlying hash function. The appeal of HMAC lies in the exact relationship between the strength of the embedded hash function and the strength of HMAC proven by the designers of HMAC.

Question: According to the birthday paradox, 2^{64} attempts have at least a 50% chance of producing a collision in MD5 (128-bit output length) and this looks feasible with today's technology. Can we use MD5 for HMAC?

Answer: Yes

To attack MD5, the attacker can choose any set of messages and work on these offline on a dedicated computing facility to find a collision. Because the attacker knows the hash algorithm and the default IV, the attacker can generate the hash code for any message chosen by the attacker.

However, when attacking HMAC, the attacker cannot generate (message, code) pairs offline as the attacker does not know the secret key K. Therefore, the attacker must observe a sequence of messages generated by HMAC under the same key and perform the attack on these known messages. For MAC, this requires 2^{64} observed blocks (272 bits) generated using the same key. On a 1-Gbps link, this requires observing a continuous stream of messages with no change in key for about 150,000 years to succeed, which makes attacking HMAC with MD5 infeasible with today's technology.

2.4 Authenticated Encryption (AE)

Authenticated encryption (AE) refers to an encryption system that simultaneously provides both confidentiality and authenticity (integrity) of communications. Authenticated Encryption (AE) is required because it provides both confidentiality and integrity in a single operation, ensuring that data remains private while also being protected from tampering. A MAC alone only guarantees integrity and authenticity but does not provide confidentiality, leaving the data vulnerable to exposure if not encrypted. Moreover, AE addresses the risk of developer mistakes when using encryption and MAC separately, such as incorrect ordering or improper implementation, which can lead to vulnerabilities. By integrating encryption and authentication, AE reduces the chances of these errors, offering a more secure and reliable solution for applications like secure messaging and financial transactions.

The four common approaches to providing both confidentiality and encryption are:

- **Hashing followed by encryption (H—>E)**:
  First, compute the cryptographic hash function over M as $h = H(M)$. Then encrypt the message plus the hashed value as $E(K, (M \parallel h))$.
- **Authentication followed by encryption (A—>E)**:
  - Requires two keys
  - First authenticate the plaintext by computing the MAC value as $T = MAC(K_1, M)$. Then, encrypt the message plus tag as $E(K_2, [M \parallel T])$.
  - This approach is taken by the SSL/TLS protocols.
- **Encryption followed by authentication (E—>A)**:
  - Requires two keys
  - First encrypt the message to yield the ciphertext as $C = E(K_2, M)$. Then, authenticate the ciphertext with $T = MAC(K_1, C)$ to yield (C, T).
  - This approach is used in the IPSec protocol.

- This approach can withstand stronger attacks.

- **Independently encrypt and authenticate (E + A)**:
  - Requires two keys
  - Encrypt the message to yield the ciphertext as $C = E(K_2, M)$. Authenticate the plaintext with $T = MAC(K_1, M)$ to yield (C, T). These operations can be performed in either order.
  - This approach is used by the SSH protocol.

3 Digital Signatures

Digital signature can be considered as the most significant development from the work on public-key cryptography. While message authentication codes (MACs) rely on a shared symmetric key between communicating parties, digital signatures utilize public-key cryptography to accomplish a similar objective.

Figure 10 depicts a generic model of the process of constructing and using digital signatures. Consider a scenario where Bob intends to send a message to Alice. Although Bob doesn't prioritize the confidentiality of the transmission, he seeks to assure Alice that the message is from him. To achieve this, Bob employs a secure hash function like SHA-512 to compute a hash value for the message. This hash value, along with Bob's private key, is inputted into a digital signature generation algorithm, resulting in a short block that serves as the digital signature. Subsequently, Bob sends the message along with the attached signature. Upon receiving the message and the signature, Alice first calculates the hash of the message and provides the hash value and Bob's public key as inputs to a digital signature verification algorithm. If the algorithm returns the result that the signature is valid,

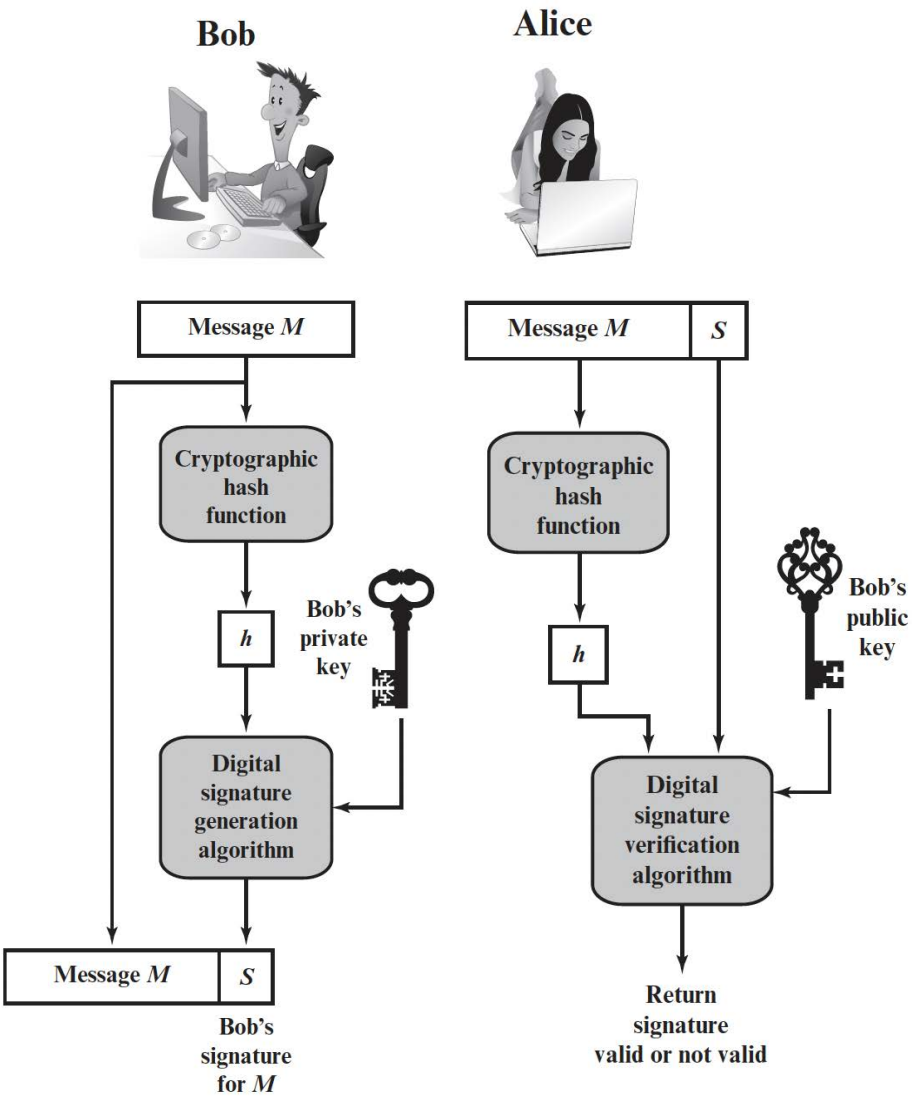
- Alice is assured that the message was signed by Bob:
  No one else other than Bob has access to his private key which can create a signature that can be verified with Bob's public key.

- Alice is assured the integrity of the message is protected:
  Without access to Bob's private key, no one can alter the message to produce a hash value that can be verified with Bob's public key.

3.1 Digital Signature Algorithm (DSA) Approach

The Digital Signature Algorithm (DSA) was proposed in 1991 and published by the National Institute of Standards and Technology (NIST) via Federal Information Processing Standard FIPS 186. DSA approach makes use of the Secure Hash Algorithm (SHA) and provides only the digital signature function. Unlike the RSA approach in Section 3.2, the DSA approach cannot be used for encryption or key exchange.

An overview of the DSA approach is provided in Figure 11a. The DSA approach uses a hash function to generate a hash code from the message M. The hash code is then provided as input to a signature function along with a random number k generated for this particular signature. The signature function also depends on the sender's private key (PR_a) and a set of parameters known to a group of communicating principals. These set parameters can be considered to constitute a global public key (PU_G). The resulting signature consists of two components, labeled s and r. At the receiving end, the hash code of the incoming message is generated. The hash code and the signature are input to a verification function that depends on the global public key as well as the sender's public key (PU_a),



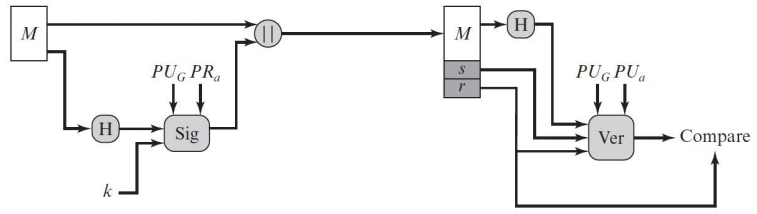
(a) Bob signs a message (b) Alice verifies the signature

Figure 10: Simplified depiction of essential elements of digital signature process
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

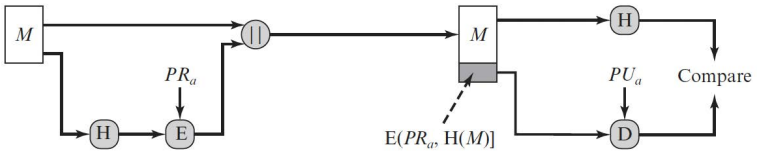
which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component r if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

3.2 RSA Approach

In 2013, an expanded version of FIPS-186, referred to as FIPS-186-4 was proposed to incorporate digital signature algorithms based on RSA. An overview of the RSA approach is provided in Figure 11b. In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because



(a) DSA approach



(b) RSA approach

Figure 11: Approaches to Digital Signatures
Source: Cryptography and Network Security (Seventh Edition - William Stallings)

only the sender knows the private key, only the sender could have produced a valid signature.

4 Practice Quiz

Question 1: Cryptographic hash functions are used to ensure confidentiality and integrity in communications. Which of the following is TRUE regarding cryptographic hash functions?

a) A cryptographic hash function must satisfy preimage resistance.
b) It is possible to find cryptographic hash functions that are collision-free.
c) Cryptographic hash functions can not provide origin authentication.
d) Cryptographic hash functions are used to ensure confidentiality and integrity in communications.

Explanation:
(a) Correct - A cryptographic hash function must satisfy pre-image resistance, second pre-image resistance, and collision resistance.

(b) Incorrect. Hash functions map longer, potentially variable-length sequences to shorter sequences. So, collisions are bound to happen because the input space is much larger than the output space.

(c) Correct - Cryptographic hash functions do not provide origin authentication. We will need some form of digital signatures or MACs for that.

(d) Incorrect - Cryptographic hash functions ensure integrity only.

Question 2: "Given a hash function H, with n possible outputs and a specific value H(x), if H is applied to k random inputs, what must be the value of k so that the probability that at least one input y satisfies H(y) = H(x) is 0.5", is a reference to the .....

a) Authentication code
b) Collision resistant
c) Big-endian
d) Birthday attack

***Explanation**: This question checked whether you know how the "birthday attack" is applicable to cryptographic hash functions. All the other answers are unrelated.*

Question 3: The MAC does not provide a digital signature because both the sender and receiver share the same key. TRUE or FALSE.

a) TRUE
b) FALSE

***Explanation**: This is true. MACs start with the premise that there is a shared key established already.*
Question 4: Confidentiality can be provided by performing message encryption ..... the MAC algorithm.

a) Before
b) Before or after
c) After
d) During

***Explanation**: This is related to the four "MAC" "encrypt" scenarios we discussed. Either you can calculate MAC and then encrypt, or you can encrypt and calculate the MAC.*

Question 5: An important characteristic of the MAC algorithm is that it needs to be reversible. TRUE or FALSE.

a) TRUE
b) FALSE

***Explanation**: Like hashes, the MAC has to be irreversible. Therefore, the statement is false.*

Question 6: To create a ....., a user calculates two quantities, r and s, that are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message H(M), and an additional integer k that should be generated randomly or pseudorandomly and be unique for each signing.

a) Signature
b) Hash authentication
c) Secret key
d) Global key

***Explanation**: The process describes the DSA digital signature algorithm. Therefore, the answer is A. We didn't cover this part in detail in the lecture. Please refer to the textbook content on the DSA algorithm. The expectation at the exam is only the high-level ideas of DSA.*

Question 7: In the digital signature algorithm, the user's ..... is represented by x, which is a random or pseudorandom integer with $0 < x < q$.

a) Per message secret number
b) Private key
c) Global key

d) Public key

Explanation*: In DSA the user's private is a random number generated by the user and is commonly denoted by x. We didn't cover this part in detail in the lecture. Please refer to the textbook content on the DSA algorithm. The expectation at the exam is only the high-level ideas of DSA.*

Question 8: A good hash function has the property that "the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random". TRUE or FALSE.

a) TRUE
b) FALSE

Explanation*: This is a desired property of a hash function, even if we do not directly stipulate it. The outputs of the hash function must appear random.*

Question 9: You are given the below string.

"5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8"

This string is in hex (i.e., 2 hex characters represent a byte).

Which hashing algorithm is the most likely to have produced this result if you suspect it is an output of a hash function?

a) SHA-1
b) SHA-224
c) SHA-256
d) SHA-512

Explanation*: Using the given information, we can calculate the length of the string. There are a total of 40 characters in the string. Two hex characters represent a byte. So, the total length is 20 bytes. That means the total length is 160 bits. Out of the hash functions we discussed, only SHA-1 results in 160-bit outputs.*

Question 10: Consider the same hash value again.

"5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8"

We already know that hash values are theoretically reversible. Rainbow tables pre-compute hash values for common words and store them as lookup tables. Use an online hash lookup table to find the actual plaintext corresponding to this hash.

Example online tool: https://sha1.gromweb.com/

What is the correct plaintext corresponding to the given hash?

a) Password
b) ChangeMe
c) password

d) querty

Explanation*: The online tool will return "password" as the plaintext. In practice, to defend against rainbow table lookups, we use salt (a random string that is concatenated with the original plain text before calculating the hash).*